This specification provides an outline for implementation of memory BIST in both hard and soft core products. This specification will act as a blueprint for all new designs within ARM with the end goal of creating a re-useable and constant memory BIST architecture and memory BIST interface for all customers (ARM internal included).

The memory BIST architecture will require functionality to exist in both hard and soft cores since it is ARM's desire to place memory test interfaces in non-critical paths. This access can be achieved by accessing write and read paths n pipeline stages away from the physical array. Physical mapping of arrays is required for best array test quality and soft cores will provide an address porting to allow the end user to re-configure address pins to match physical implementation once it is known.

ARM will provide an optional memory BIST controller in soft core implementations and will provide a custom implementation with hard cores (not optional). It is clear that every end customer has their own set of beliefs when it relates to memory BIST algorithms. It is ARM's goal to provide standard and custom algorithms which test for numerous fault classes and types. While these patterns may not perfectly match every end user's test methodology, they should be sufficiently robust such that the end user is not needing to create a new implementation.

There are two interfaces receiving special attention in this specification. First is the user interface. The user interface is specified with future expansion in mind. Use from core to core will remain the same, as will the primary instruction register that is loaded by this interface. This provides the same look and feel across the product line. The second interface exists at the memory access points. It is important to have a common interface among various cores to insure that the end user's own

BIST solution can be configured for all future ARM cores with a common interface as well as meeting requirements for our internal BIST engine.

This document does not address specific ATPG test needs such as array bypass or controls but does include a couple of features that are required regardless of ATPG testability design style.

MBIST Architecture

ARM memory BIST strategy revolves around the ability to run testing with physically mapped arrays in order to leverage pattern effectiveness. Hard cores are always physically mapped in X (rows/distance to/from sense amp) and Y (column or sense amp selection) dimensions. Soft cores will have documented mappings of the memory test interface to the logical address provided at the core to memory interface. It will be a required exercise of the integrator (if they are to realize maximum test effectiveness) to either :

1) perform the physical mapping by scrambling the BIST_dispatch unit for each interface to match intended mapping
2) Force physical mapping in compiled rams to match the predefined ARM mapping

The Memory BIST architecture will be centered around a single memory BIST test engine which distributes control signals to all arrays in the core. It is assumed that one memory BIST implementation will be created per core although it may be possible to integrate an engine across multiple cores. Providing two memory BIST engines per core is allowed but tester to memory BIST controller test interface must be managed and selected by the system integrator. Test signal distribution and routing impact is minimized by the use of a 4 bit dataword bus leaving the memory BIST controller. Dataword expansion and testing will occur at a specialized unit called mBIST_dispatch which acts as glue logic between the memory BIST controller

and the memory BIST test interface. This glue logic performs address scrambling (mapping to physical space), specialized pipelining (local to targeted arrays), and dataword comparisons and test fail datalogging. This significantly reduces the excessive routing overhead usually associated with a centralized BIST controller.

The dispatch unit containing glue logic provides design specific implementations to be accommodated while enabling the memory test interface and BIST controller to be a fixed standard from design to design. End user test interface can be a standard from design to design by having a BIST controller that is off the shelf re-usable in all future ARM cores.

Memory BIST is launched by configuring the core's test interface to allow access to the memory BIST interface. Test algorithm progress (measured by address expire flags) and pass/fail status are continuously delivered on a 4bit output only data bus (ported to macrocell). Further datalogging information can be obtained by shift access (using same external BIST interface) of the BIST controller or failing unit's dispatch unit. The amount of datalogging and trace support in the dispatch unit will be design dependant and may be configurable.

Each memory BIST interface may support multiple arrays. Each of these arrays will be supported as separate arrays in the BIST instruction register and failure log. During test execution , these arrays will also be tested by appending their enable lines to the Yaddr space. A BIST dispatch unit can accommodate mapping of enable lines to higher address bits if so enabled by the instruction register's enable lines. Note that this does not fit the usual definition of serial test or parallel test of these target arrays but the arrays are instead treated as segments of a larger whole array.

Centralized BIST controller and dispatch units:

The MBIST User interface (test access to BIST Ctl) is illustrated in Figure 6.

The MBIST test interface is defined by the below table. Of concern is the access mechanism for shift loading of the test configuration. This shift process will be performed on the core clock domain. As cores progress into the future, this may require a pll switching mechanism to allow slow manual shift (due to tester interface limitations) and a switch to high speed pll clocking in order for memory test to occur at speed (if clock rate exceeds the tester cycle or if pad timing limitations suppress maximum clocking speeds). It may make sense to implement a divide by four shifting clock for Jaguar in order to support future migration paths.

Test Interface to MBIST		
Core Port Signal	I/O , Size	Description
MBIST_RUN	In	Memory testing launched by this signal
MBIST_SHIFT	In	Enables control register data load
MBIST_DIN	In	Provides serial load of control registers
SCANMODE	In	Disables overrides at dispatch units
MBIST_DOUT	Out , <3:0>	Provides runtime status <done, fail, Xexpire, Yexpire>

The test configuration is loaded into a 32 bit register by asserting MBIST_SHIFT for 32 clocks and delivering LSB first, MSB last. MBIST_RUN MUST be inactive during this register load. Execution of register load terminates when MBIST_SHIFT is de-asserted. Assertion of MBIST_RUN will begin the memory test selected and test termination is signalled by the MBIST_DOUT<3> bit. MBIST_DOUT<1:0> will toggle during test to mark algorithm progress. MBIST_DOUT<2> or the fail bit will be sticky.

MBIST_SHIFT will be gated by MBIST_RUN to prevent unknown hazards in the engine.

MBIST_DOUT<0> doubles as the shift data out path of the instruction register whenever (MBIST_SHIFT & ~ MBIST_RUN).

MBIST Instruction Register

Please refer to Figure 7.

Array Enables and Flags <39:24>

Each tested array in the core will be assigned an enable bit as specified to the TRM document. This implementation is defined in this manner to account for the desire to test all arrays in parallel and to also allow flexibility to test arrays individually should the need arise. One example is the need to test only 50% of the arrays per pass due to maximum power concerns. Another possibility may include initialising different arrays to different background patterns for use in ATPG or in iddq. An array is not defined by the number of memory test interfaces but is defined by the number of individually tested arrays (note that a single memory test interface may have multiple array enables).

This portion of the instruction register also serves a second purpose. The outside user sees a single failure flag on the external interface. Scanout of this portion of the register , will reveal the P/F status of each array to further identify the failure array.

maxYaddr <23:20>

The maxYaddr field allows the user to specify the maximum Yaddr (a.k.a. column addr) to be used during test. This is especially useful when core derivatives of with varying cache sizes are employed or when different sized arrays within the core are to be tested. This bist controller architecure does not prevent multiple arrays of different sizes from being tested at the same time. To perform test of such scenarios, the user programs a maximum address size equal to the largest array and the dispatch unit blocks out of range addresses(by gating RE,WE) for the smaller array.

The following table describes the register settings and resulting address sizes :

MaxYaddr	Size	BistYaddr active bits [11:0]
0000	0	000000000000
0001	2	000000000001
0010	4	000000000011
0011	8	000000000111
0100	16	000000001111
0101	32	000000011111
0110	64	000000111111
0111	128	000001111111
1000	256	000011111111
1001	512	000111111111
1010	1k	001111111111
1011	2k	011111111111
1100	4k	111111111111
>1100	--	Not supported

maxXaddr <19:16>

Similar to the Y address , the X address field slects the maximum number of addressable rows. Row depths beyond 512 are usually impractical due to bitline RC's but a space of up to 2k is supported here if the user wishes to map planar features in the topology to the Xaddr space (we recommend that planar or block addresses be mapped on top of Y address space, else bitline stress tests based on Xaddr space may lose effectiveness).

MaxXaddr	Size	BistXaddr active bits [10:0]
0000	0	00000000000
0001	2	00000000001
0010	4	00000000011
0011	8	00000000111
0100	16	00000001111
0101	32	00000011111
0110	64	00000111111
0111	128	00001111111
1000	256	00011111111
1001	512	00111111111
1010	1k	01111111111
1011	2k	11111111111
11xx	--	Not supported

Engine Control <15:10>

The engine control portion of the register is used to specify special state machine behaviors such as stop on error. An <15:10> = 0 setting specifies default behavior.

<15:10>	Behavior	Description
00 0000	Default	Test runs to completion
1x xxxx	Real Time Fail bit	Fail bit is real time, sticky fail appears at end of test
xx 0001	Stop on fail	Stops on fail with <16 cycle latency
xx 0010	Stop at end of algorithm if fail	Stops on fail after current algorithm "N" is completed
00 1110	Load dispatch unit instruction	Scan dispatch unit as selected by one hot <31:16>
00 1111	Read dispatch unit datalog	Scan dispatch unit as selected by one hot <31:16>

The real time failure bit is provided for lab use should the test/lab personnel wish to monitor soft fail behaviors.

Stop at end of algorithm pass is provided as a means to allow testing to progress until end of a failing algorithm segment. This may be useful for bitmapping if the dispatch unit provides scannable test access mechanisms for downloading all array entries.

Scanning of dispatch unit will place selected dispatch unit scan ports between MBIST_DIN and MBIST_DOUT when MBIST_RUN is asserted. Access is disabled once MBIST_RUN is de-asserted. Any operation loaded into dispatch unit will be executed at time of MBIST_RUN de-assertion. (Note : I need to better define handshake to allow maximum efficiency and also proper function as mBIST_run will require warm-up cycles and must allow enough time for the operation to complete, would probably be ok if I delayed internal override releases by appropriate cycle counts).

Dispatch units are loosely defined at this time and will be built to core's need. They should provide minimum datalogging information, such as first fail address and fail bit. This information should be accessible by the dispatch datalog scan operation. Use of scannable dispatch unit instructions is optional.

Pattern/Algorithm selection <9:4>

Numerous standard patterns and ARM created bitline stress patterns are provided for use in test. Algorithms are created in pieces to allow for maximum flexibility in unknown final test design. Areas of test supported include go/nogo, iddq , data retention, ATPG, gross function test, and stress testing. The following table describes supported patterns and later paragraphs describe their use. Use of segmented testing can be used to maximize softbin and yield tracking information in the fab and to provide the end test design to best match the fab's memory test own beliefs .

<5:0> selection	Name	"N"	Description
00000	WriteSolids	1N	Write specified dataword into all entries
00001	ReadSolids	1N	Read dataword from all entries
00010	WriteCkbd	1N	Data or databar selected by X<0> ^ Y<0>
00011	ReadCkbd	1N	Data or databar selected by X<0> ^ Y<0>
00100	March C+	14N	Increment decrement XY fast RWR march
00101	PttnFail	6N	Tests BIST failure detection
00110	RW_Xmarch	6N	Increment decrement wordline fast march
00111	RW_Ymarch	6N	Increment decrement bitline fast march
01000	RWR_Xmarch	8N	Increment decrement wordline fast march
01001	RWR_Ymarch	8N	Increment decrement bitline fast march
01010	Bang	18N	Increment decrement wordline fast bitline stress
11111	Go / No Go	30N	2*W/RCkbd, RWR Ymarch, X Bang

Definitions:

column --> dimension in array parallel to bitlines (on the same SenseAMP)
row --> dimension in array parallel to wordlines
row fast / Xfast --> target cell moves along bitlines before moving to next column
col fast / Yfast --> target cell moves across bitline pairs before row/wordline
Xfast Increment --> target cell begins nearest sense amp, moves away
Xfast Decrement --> target cell begins furthest point from sense amp, moves closer
Yfast Increment --> Yaddr space moves from 0 to max, east-west relationship
Yfast Decrement --> Yaddr space moves from max to 0, opposite of increment
N --> Number of addressable entries

BIST Pattern Specification:

All memory BIST tests are executed against physically mapped address space. This simply means that the least significant Xaddr switches between adjacent rows (LSB+1 switches between every 2nd row, and so on). Yaddr space is also physically mapped. This allows for efficient and direct targeting of memory faults with the supplied patterns.

"***writeCkbd***" is performed Xfast. This pattern is 1N, writing only. Data polarity set by xor(Xaddr[0],Yaddr[0]).

"***ReadCkbd***" is performed Xfast. This pattern is 1N, reading only. Data polarity set by xor(Xaddr[0],Yaddr[0]).

"***writeSolids***" is performed Xfast. This pattern is 1N, writing only. Data polarity = true.

"***ReadSolids***" is performed Xfast. This pattern is 1N, reading only. Data polarity = true.

"***MarchC+***" is performed Xfast with the following sequence:

1) write solids (initialize array) incr
2) Rdata, Wdatabar, Rdatabar incr
3) Rdatabar, Wdata, Rdata incr
4) Rdata, Wdatabar, Rdatabar decr
5) Rdatabar, Wdata, Rdata decr
6) Rdata verify decr

"***RWR_Xmarch***" is performed Xfast with the following sequence...

1) write solid (initialize array)
2) Rdata/Wdatabar/Rdatabar inc
3) Rdatabar/Wdata/Rdata decrement
4) Rdata solid

"***RWR_Ymarch***" has the same sequence as RWR_Xmarch but is performed Yfast.

"***RW_Xmarch***" is performed Xfast. This 6N pattern has the following sequence...

1) write solid (initialize array)
2) Rdata/Wdatabar inc
3) Rdatabar/Wdata decrement
4) Rdata solid

"***RW_Ymarch***" is 6N and performed Yfast with the same sequence as RW_Xmarch.

"***pttnFail***" is performed Xfast. Executes a writeSolid pattern followed by a ReadSolid. Fails are injected by reversing data polarity on select addresses during ReadSolid. This pattern is REQUIRED to insure BIST detection logic (at target array) is functional.

"***bang***" , 18N, is performed Xfast, executing consecutive multiple writes and reads on a bitline pair.

1) write solid (initialize array)
* 2) read data target, write databar target, 6 * write databar row0, increment to next target Xfast
** 3) 5 * read databar target, write data row 0, read databar target, write data target
4) read data (verify array)

* This segment "bangs" bitline pairs insuring proper equalization after writes. Insufficient equalization/precharge will result in a slow read when opposite data is read from the same bitline pair. Slow reads in self timed caches result in functional failure not found in popular "single shot" algorithms like March C.

** This segment attempts to walk down a bitcell, write opposite data on that bitline pair, read target cell data. This failure mechanism is less common in 6T ram cells (vs. 4T or DRAM).

*,** Use of the "sacrificial" row, also helps detect open decoder faults in the Xaddr space (Yaddr not subject to fault class architecturally because of smaller decode blocks) in absence of graycode pattern sequences.

*,** This pattern does detect stuck at fault, but its primary intent is to address the memories' analog characteristics.

"***go/nogo***", 30N is the ARM test suite selected for memory tests should the end user not desire to implement their own test strategy. Selection of this pattern will result in the following tests to be executed as listed below. This provides the end user a pass fail status for the array. The below test suite provides a comprehensive functional test of the arrays (data retention and other stress tests not included). The series of tests in gonogo are the culmination of internal ARM memory test engineers' past experiences in memory test.

1) Perform wckbd 5's
2) Rckbd 5's
3) Wckbd a's
4) Rckbd a's
5) RWR_Ymarch 6's
6) Bang 0's

DataWord <3:0>

At test instruction load, the user specifies the root dataword used during the test algorithm (exception during go/nogo BIST algorithm). This allows the user to select values stored into initialised arrays (iddq or ATPG) or to select new datawords to search for unexpected sensitivities during march and bitline stress tests.

MBIST Memory Interface

The memory interface is defined in the below table. Its use is required for all memory devices on a core, regardless of whether or not ARM provides the memory for that core. This will allow the user to make use of the ARM BIST controller without special modifications. ATPG related support is also defined here.

Signal	I/O , Size	Description
MTestSel	In , <?:0>	Optional, Used for serial test of multiple downstream arrays
MTestWEx	In	Dictates write operation
MTestREx	In	Dictates read operation
MTestADDR*	In, <maxAddr:0>	Target Address
MTestDINx	In, <datawidth:0>	BIST Write Data
MTestDOUTy	Out, <datawidth:0>	BIST read Data
MTestON	In	Global Override of cache logic
ArrayDisable	In	Blocks all array activity , regardless of operation mode

* May have single Address bus if mBIST_dispatch handles all scrambling concerns. This is still being thought out.

Memory BIST Interface Definitions

"MTestSel<?:0>"

Should a memory BIST interface support more than one downstream array, separate enables are required to select array being targeted. This signal may also be required for a single array that has multiple repairable areas. A single array interface does not require use or porting of this signal.

"MTestWEx"

This signal (if not blocked internally by MTestSel or MTestON) indicates a write operation is being delivered to the memory test array interface. Address and Data information will be delivered coincident with this signal. If a write operation takes two cycles to complete , this signal is designated "MTestWE2". If multiple arrays are supported by the interface, all with varying pipeline delays, the x designation shall be used ("MTestWEx"). Use of the pipeline delay length in the signal name allows for easier BIST integration by the BIST interface unit or end users own implementation.

"MTestREx"

Read enable for array under test. See MTestWEx signal for further information.

"MTestADDR<?:0>"*

Address being targeted in array under test. Signal is as wide as maximum number of addressable elements in targeted arrays.

"MTestDINx, MTestDOUTy"

Data in and data out ports for BIST operations. MBISTDOUT delivered "x+y" cycles later as defined by MBISTREx. Example: An MBISTRE3 specifies that the read operation will occur 3 clocks after MBISTRE3 information is asserted. The dataout pipeline may have its own offset and is represented by "y". If multiple arrays are supported by a single interface, and some arrays have smaller data widths than the maximum array width, the memory BIST interface shall be required to deliver "0" on all unused data bits for all read operations. If datain and dataout busses have varying pipeline offsets depending on downstream array, they shall have an offset designated by "x".

"MTestON"

This is a global override signal, informing the interface and downstream logic that BIST operations are imminent. This signal is the second highest priority signal in cache logic , behind ScanMode/ScanEn. Normal function / mission mode operation is a LOWER priority operation.

"ArrayDisable"

ATPG signal , blocking writes and reads to all arrays whenever active. Blocking must occur after all scannable state elements to insure safety of array's initialized data. This signal is generated in the memory BIST dispatch unit and is the "or'd" function of IDDQtest and ScanEn. IDDQtest used to insure preservation of state during iddq scan and capture, scanEn used to preserve state during ATPG load/unload should ATPG ram tests be implemented.

* May have single Address bus if mBIST_dispatch handles all scrambling concerns. This is still being thought out.

MBIST Interface Design Rules

- MTestON overrides all normal function operations.
- MTestON is asserted at least 4 cycles prior to first operation and not released until after last operation completes. Remains static "on" for duration of BIST test.
- ArrayDisable blocks all write/read activity, regardless of device mode (functional, ATPG, BIST,etc.) and occurs after last scannable state elements.
- A BIST no-op (no operation) can and will occur. Defined as BIST cycle where RE/WE are both inactive
- Back to back operations must be supported and expected in any combination of R/W/Addr

- All operation data provided from F/F's and read data will be sampled into F/F's. (mBIST is a flop based design)
- All BIST operations can occur at full speed.
- Addr and RE/WE signals are provided in same clock cycle. DataIn may be injected at different stage if required.
- All RE/WE/Data signals must have cycle suffix designator as specified in signal descriptions.
- All read operations must be completed within 8 cycles from RE assertion to dataout. (pipeline injection points must be reasonable).
- If multiple arrays are supported by interface, address and data adjustments will occur in dispatch unit (example: addressed array is smaller than BIST_engine.....BIST_dispatch will block RE/WE for out of range addresses)
- All Variable sized arrays must be right justified in addressing wrt to incoming BISTADDR where rightmost bit is LSB.
- ONLY ArrayDisable and MTestON can be assumed to toggle without regard to operation setup/hold times. All other control signals at interface must be considered unknown from cycle to cycle.
- No address scrambling allowed between mBIST interface and actual array interface. Scrambling will occur in the dispatch unit.
- Physical mapping characteristics must be maintained on all arrays downstream of each unique array interface. (Xaddr LSB always selects between adjacent rows regardless of array supported by gi9ven interface).

MBIST Dispatch Units

A dispatch unit is required between every BIST controller to memory test interface . This block serves as glue logic between a standardized front end BIST controller and a standardized memory test interface. It also provides manual debug accesses to the array and performs data compare and datalogging of the read data from the test interface.

Dispatch Unit interface

The following table describes the incoming and outgoing interface signals of the mBIST_dispatch unit:

Input/O Data from Memory BIST controller			I/Output to memory test interface	
Signal	Description		Signal	Description
ArrayEn<?:0>	Enables target arrays (instr reg)		MTestSel	Array enables to interface
BISTTest	Game on		MTestWEx	Array write enable
MBISTData<3:0>	Write/read data		MTestREx	Array Read Enable
MBISTXaddr<?:0>	BIST Controller Wordline Addr		MTestADDR*	Array cell select
MBISTYaddr<?:0>	BIST Controller bitline Addr			
MBISTWE	BIST Controller write enable		MTestDINx	Array Data in (full width)
MBISTRE	BIST Controller read enable		MTestDOUTy	Array Data out (full width)
MBISTShiftI	Shiftenable for dispatch instr		MTestON	Array test override
MBISTShiftD	Shiftenable for datalog		ArrayDisable	Array access blocking
DispatchDin	Shift data in			
DispatchDout	Shift data out			
IddqTest	IddqTest mode			
SE	Scan enable input			
SCANMODE	ATPG testing mode			
MBISTFail<?:0>	Fail bits sent to main controller			

Notes:

1. BISTData<3:0> is linearally expanded across full memory bit width. Pipelining of this data inside of dispatch unit is done as <3:0> and expanded as needed. This expanded word is used to drive or compare the full bus width of the memory under test.
2. DispatchDout follows LSB of dispatch unit's instr or datalog register (as selected by shiftenable)
3. The memory test interface shall be driven by the dispatch unit and ONLY the dispatch unit for all defined signals (no direct connect to BIST controller as dispatch unit pipelines controller data for timing safety).
4. Support of instruction register capability inside dispatch is design specific. Datalogging/Repair information is generated and maintained in dispatch unit and is design implementation specific.

5. Multiple enable selected arrays can be tested simultaneously by mapping enables to higher order Yaddress. This is highly desirable for go/nogo testing.

Address Scrambling

Hard cores will have known physical to logical mapping of address spaces and will need to be scrambled accordingly in the BIST_dispatch unit. This is performed at the dispatch unit as address scrambling requirements may differ for different memory test interfaces.

Softcores and compiled rams will not have known addressing up front and scrambling cannot be performed in rtl, especially if the core design is synthesized prior to compilation of memories. Scrambling will be required to insure physical mapping exists with the ARM BIST algorithms. This is accomplished by adding 2 additional ports of the TestADDR to the dispatch unit. These ports represent the BIST controller address and the second represents the scrambled version of address going to the MtestAddr interface. These ports are brought up to the highest hardened level of the design and scrambling occurs by stitching the controller address to the test address as required. This stitching becomes the responsibility of the core integrator.

Although illustrative embodiments of the invention have been described in detail herein with reference to the accompanying drawings, it is to be understood that the invention is not limited to those precise embodiments, and that various changes and modifications can be effected therein by one skilled in the art without departing from the scope and spirit of the invention as defined by the appended claims.